CS 267 Applications of Parallel Computers

Lecture 11:

Sources of Parallelism and Locality (Part 3)

Tricks with Trees

David H. Bailey

Based on previous notes by Jim Demmel and Dave Culler

http://www.nersc.gov/~dhbailey/cs267

Recap of last lecture

- **ODEs**
  - **Sparse Matrix-vector multiplication**
  - **Graph partitioning to balance load and minimize communication**
- **PDEs**
  - **Heat Equation and Poisson Equation**
  - **Solving a certain special linear system T**
  - **Many algorithms, ranging from**
    - **Dense Gaussian elimination, slow but very general, to**
    - **Multigrid, fast but only works on matrices like T**

Outline

- Continuation of PDEs
  - What do realistic meshes look like?
- Tricks with Trees

Partial Differential Equations

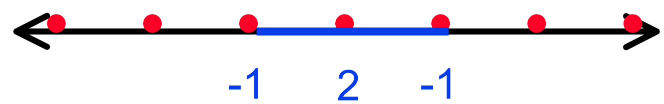
PDEs

Poisson’s equation in 1D

- Solve Tx=b where

$$
T = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}
$$

Graph and “stencil”

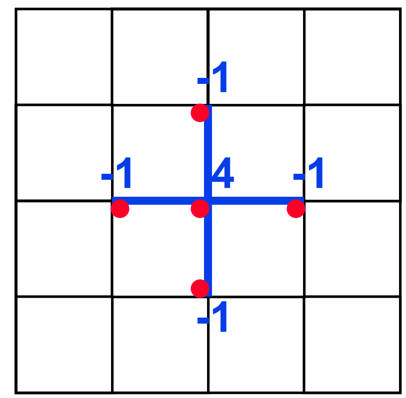


Poisson's equation in 2D

° **Solve Tx=b where**

$$
T = \left(\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
& -1 & 4 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
& -1 & & -1 & 4 & -1 & & -1 & \\
& & -1 & & -1 & 4 & & & -1 \\
\hline
& & & -1 & & & 4 & -1 & \\
& & & & -1 & & -1 & 4 & -1 \\
& & & & & -1 & & -1 & 4
\end{array}\right)
$$

Graph and "stencil"



° **3D is analogous**

Algorithms for 2D Poisson Equation with N unknowns

Algorithm	Serial	PRAM	Memory	#Procs
° Dense LU	N^3	N	N^2	N^2
° Band LU	N^2	N	$N^{3/2}$	N
° Jacobi	N^2	N	N	N
° Explicit Inv.	N^2	$\log N$	N^2	N^2
° Conj.Grad.	$N^{3/2}$	$N^{1/2} * \log N$	N	N
° RB SOR	$N^{3/2}$	$N^{1/2}$	N	N
° Sparse LU	$N^{3/2}$	$N^{1/2}$	$N * \log N$	N
° FFT	$N * \log N$	$\log N$	N	N
° Multigrid	N	$\log^2 N$	N	N
° Lower bound	N	$\log N$	N	

PRAM is an idealized parallel model with zero cost communication

Reference: James Demmel, Applied Numerical Linear Algebra, SIAM, 1997.

Mflop/s versus Run Time

- Problem: Iterative solver for a convection-diffusion problem; run on a 1024-CPU NCUBE-2.
- Reference: Shadid and Tuminaro, SIAM Parallel Processing Conference, March 1991.

Solver	Flops	CPU Time	Mflop/s
Jacobi	3.82×10^{12}	2124	1800
Gauss-Seidel	1.21×10^{12}	885	1365
Least Squares	2.59×10^{11}	185	1400
Multigrid	2.13×10^{9}	6.7	318

- Which solver would you select?

Relation of Poisson's equation to Gravity, Electrostatics

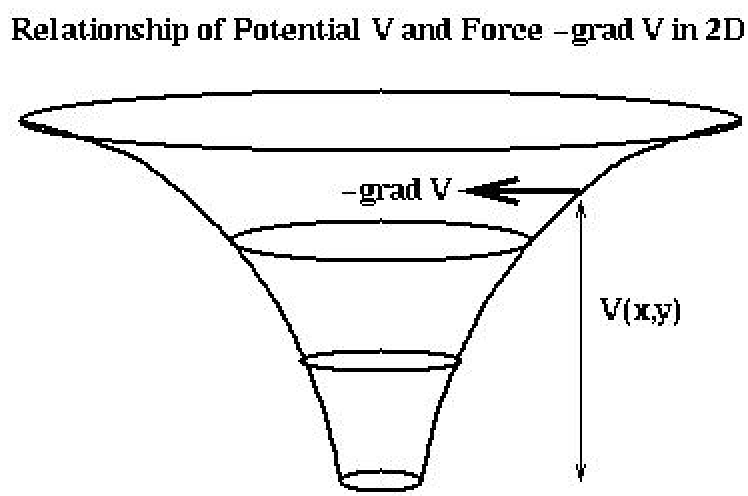
- Force on particle at (x,y,z) due to particle at 0 is

  -(x,y,z)/r^3, where r = sqrt($x^2 + y^2 + z^2$)

- Force is also gradient of potential V = -1/r

  = -(d/dx V, d/dy V, d/dz V) = -grad V

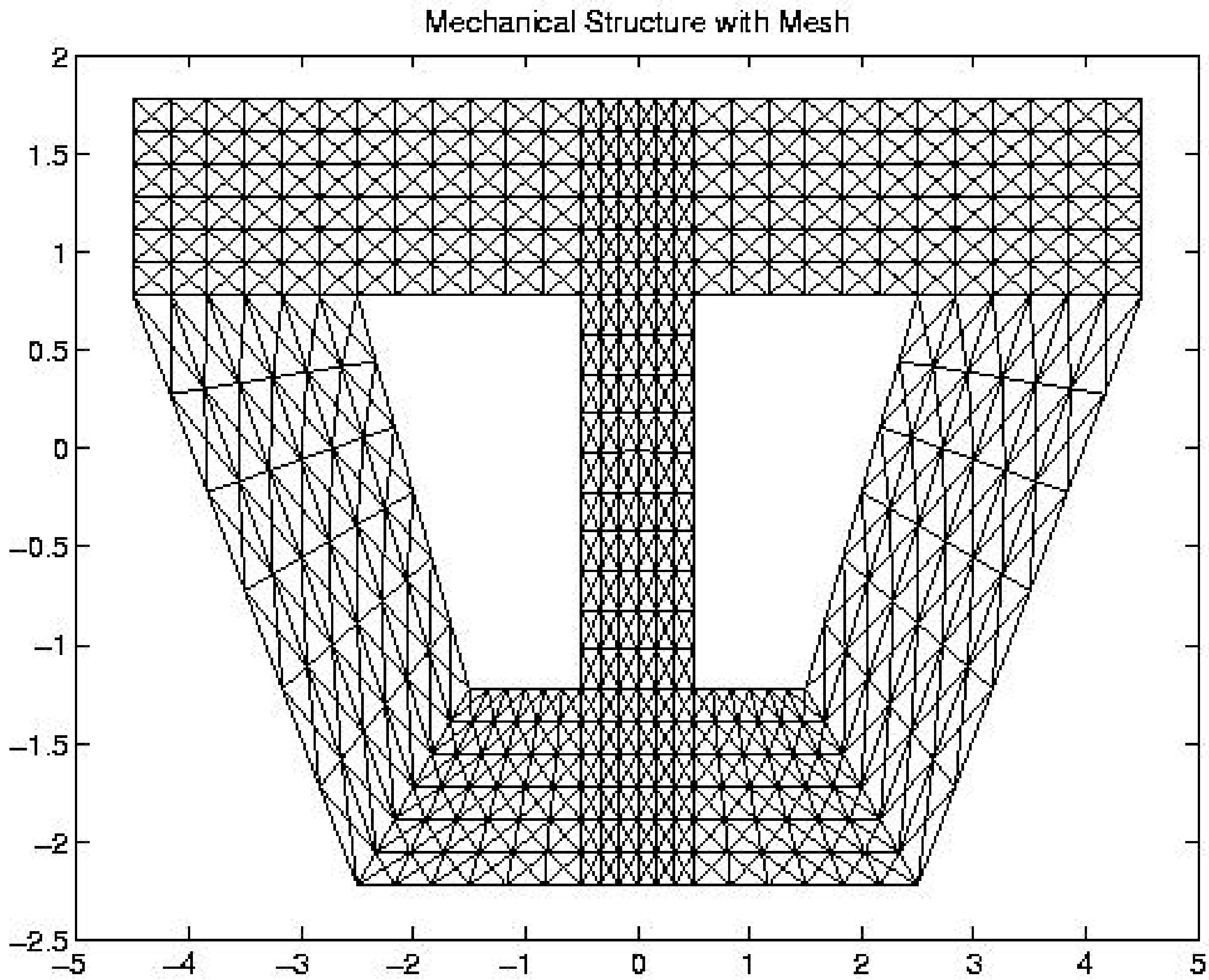
- V satisfies Poisson's equation (try it!)



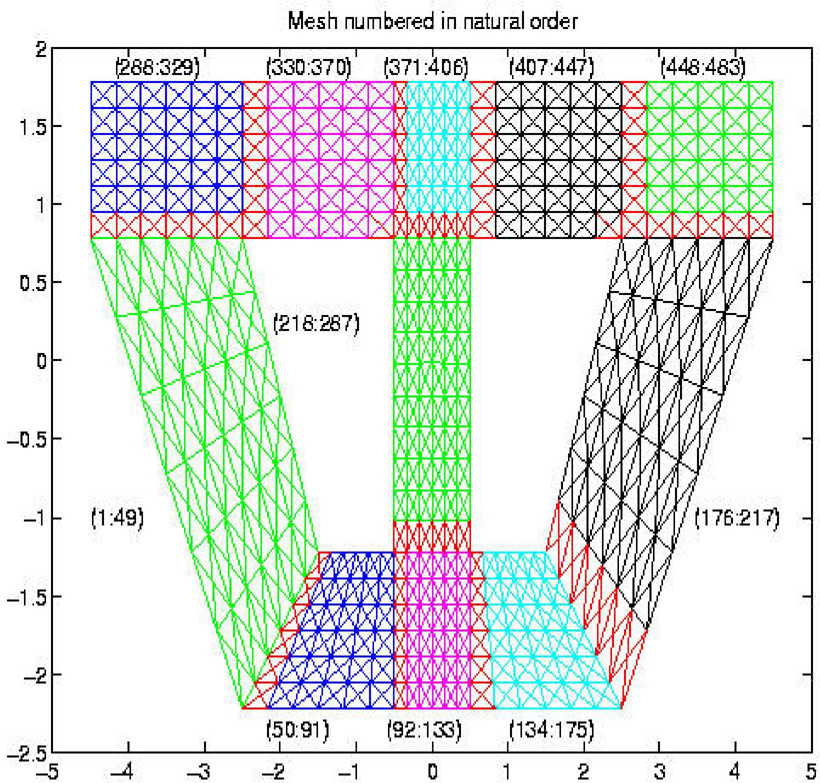
Comments on practical meshes

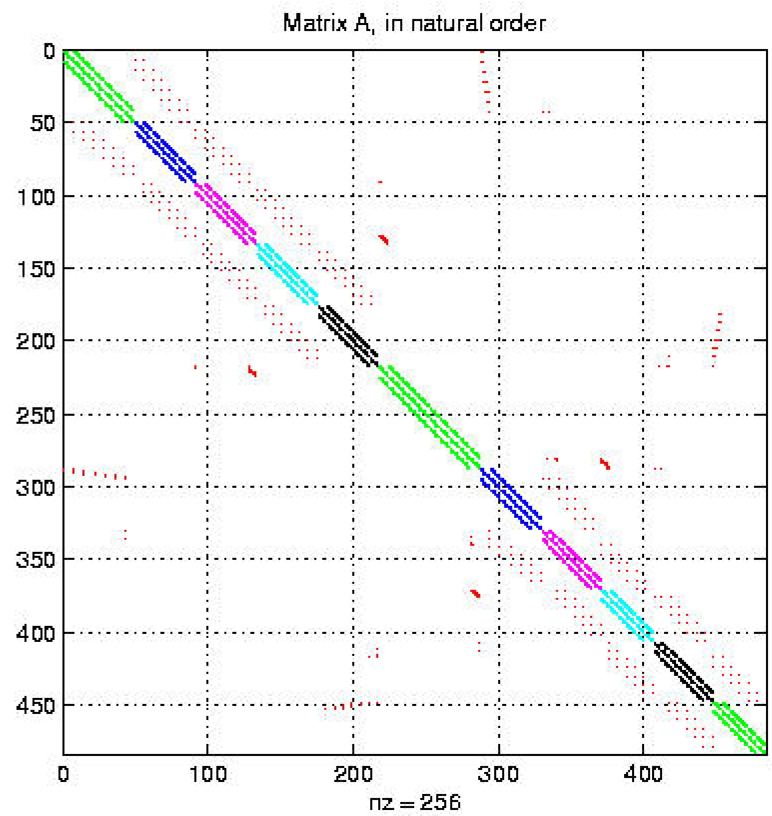
- Regular 1D, 2D, 3D meshes
  - Important as building blocks for more complicated meshes
- Practical meshes are often irregular
  - Composite meshes, consisting of multiple “bent” regular meshes joined at edges
  - Unstructured meshes, with arbitrary mesh points and connectivities
  - Adaptive meshes, which change resolution during solution process to put computational effort where needed

Composite mesh from a mechanical structure

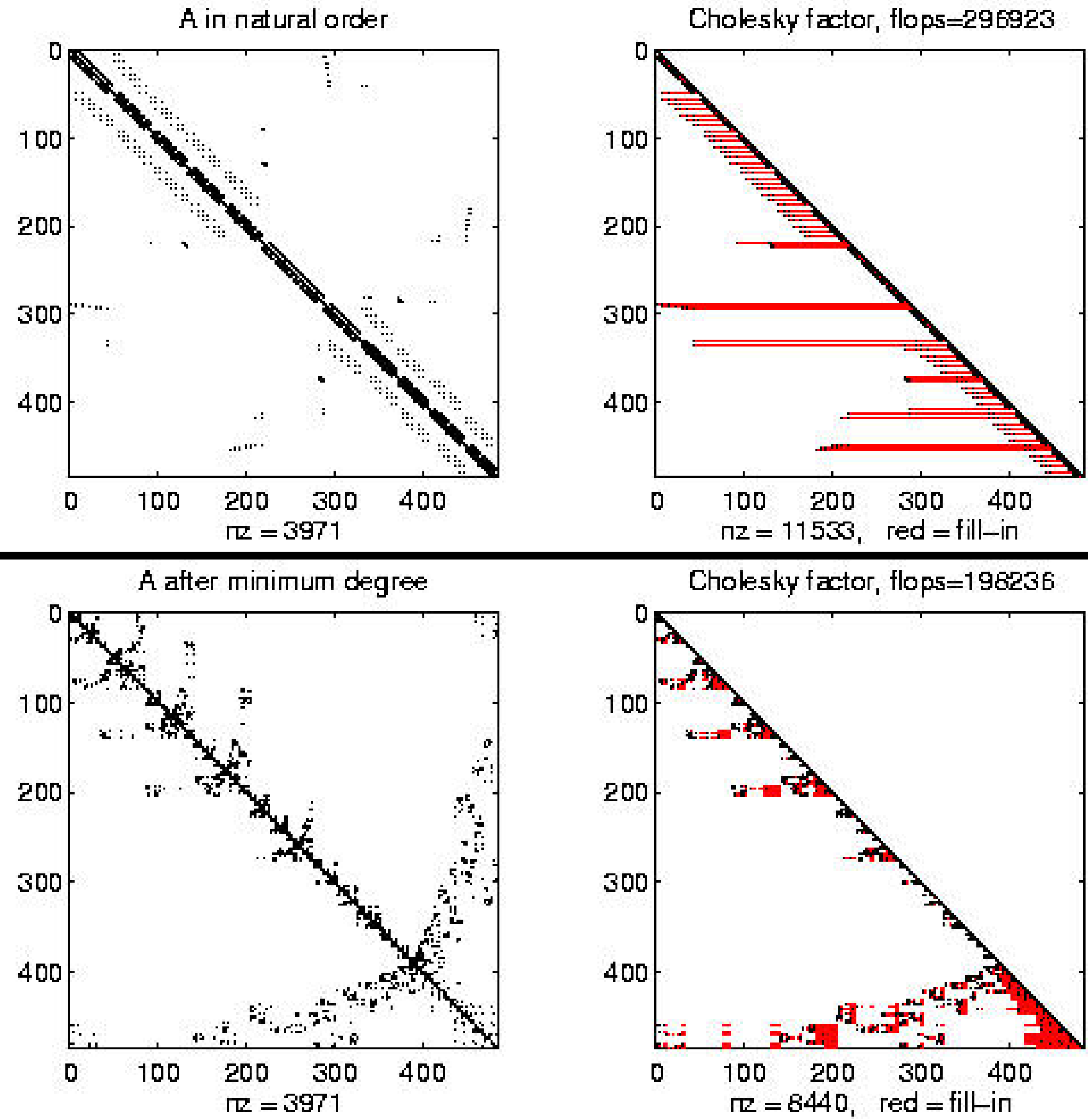


Converting the mesh to a matrix

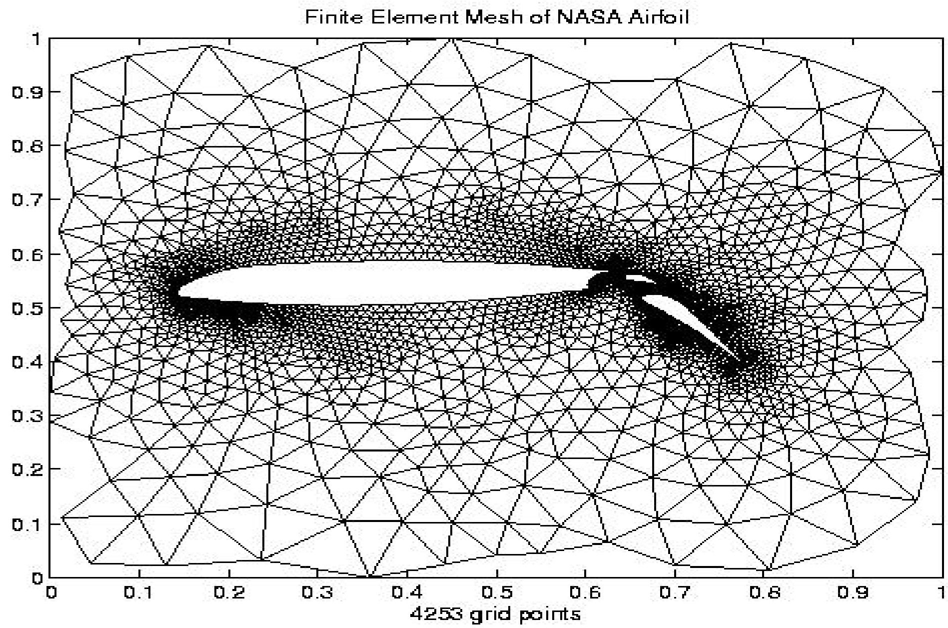


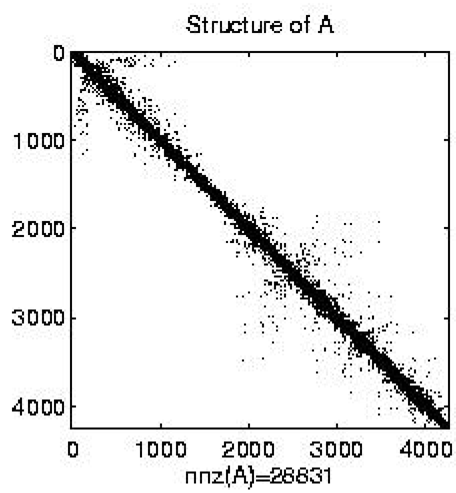


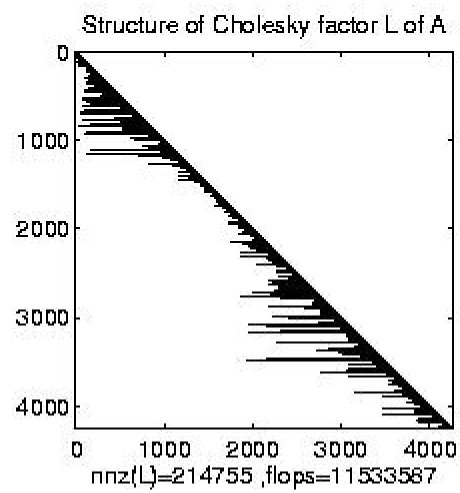
Effects of Ordering Rows and Columns on Gaussian Elimination



Irregular mesh: NASA Airfoil in 2D (direct solution)







Irregular mesh: Tapered Tube (multigrid)

Example of Prometheus meshes

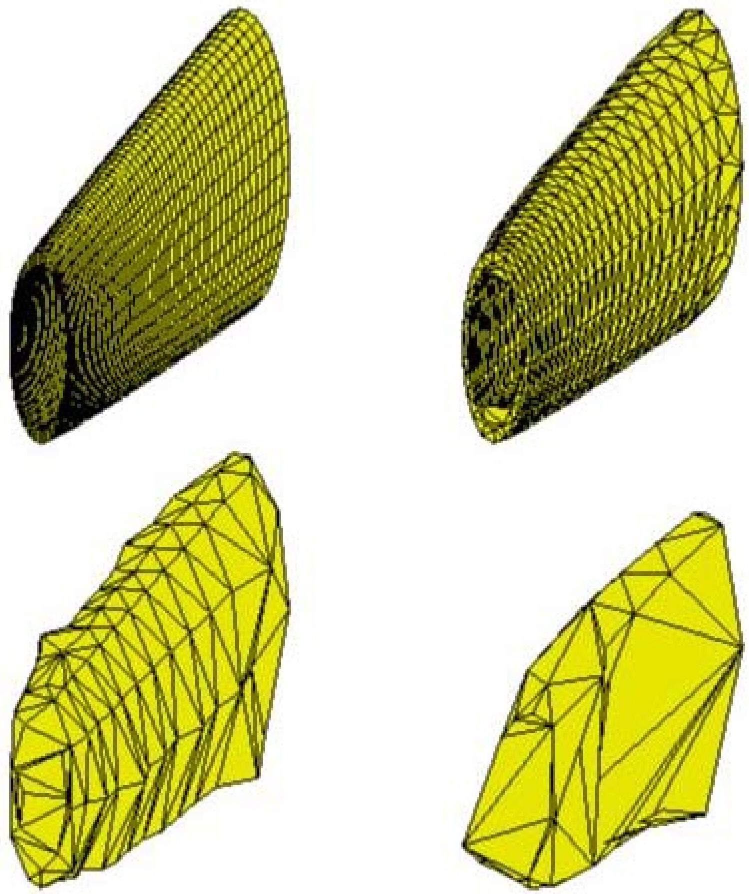
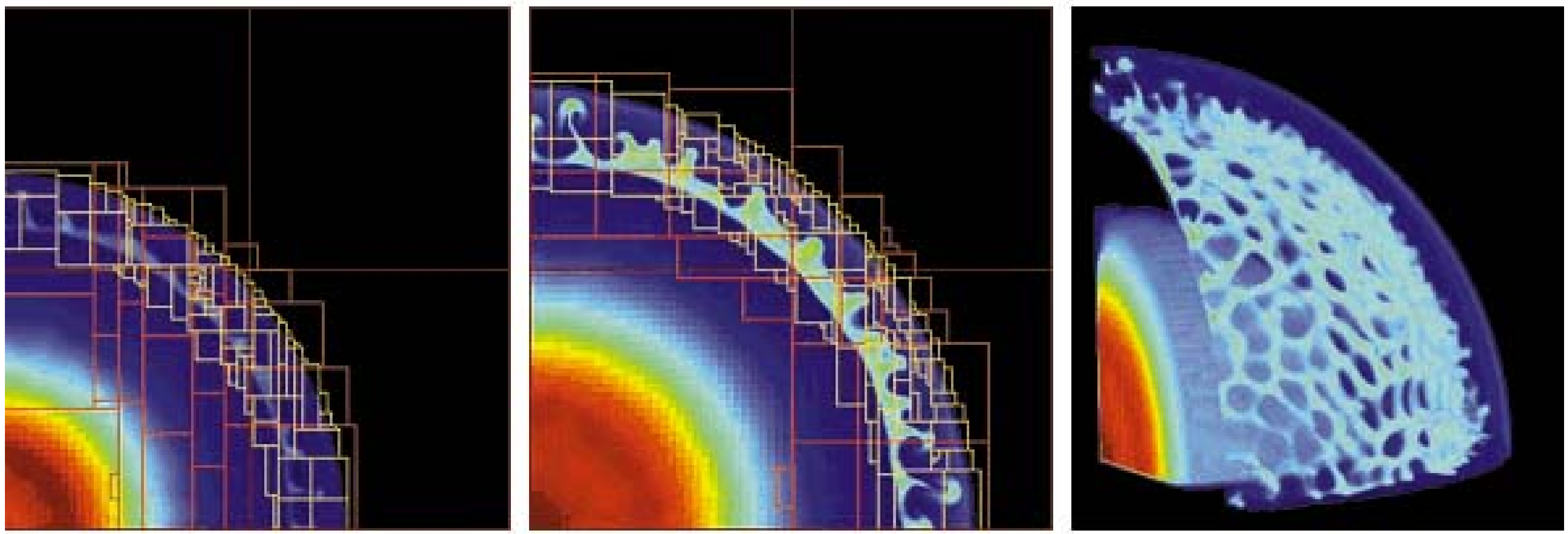


Figure 6: Sample input grid and coarse grids

Adaptive Mesh Refinement (AMR)



- Adaptive mesh around an explosion
- John Bell and Phil Colella at LBL (see class web page for URL)
- Goal of Titanium is to make these algorithms easier to implement in parallel

Challenges of irregular meshes (and a few solutions)

- **How to generate them in the first place**
  - Triangle, a 2D mesh partitioner by Jonathan Shewchuk
  - 3D harder!
- **How to partition them**
  - ParMetis, a parallel graph partitioner
- **How to design iterative solvers**
  - PETSc, a Portable Extensible Toolkit for Scientific Computing
  - Prometheus, a multigrid solver for finite element problems on irregular meshes
  - Titanium, a language to implement Adaptive Mesh Refinement
- **How to design direct solvers**
  - SuperLU, parallel sparse Gaussian elimination
- **These are challenges to do sequentially, the more so in parallel**

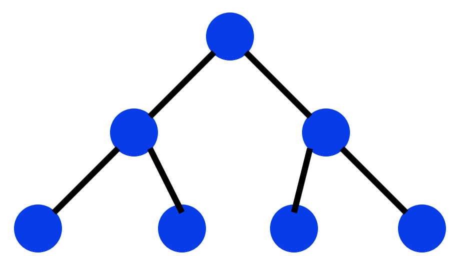
Tricks with Trees

Outline

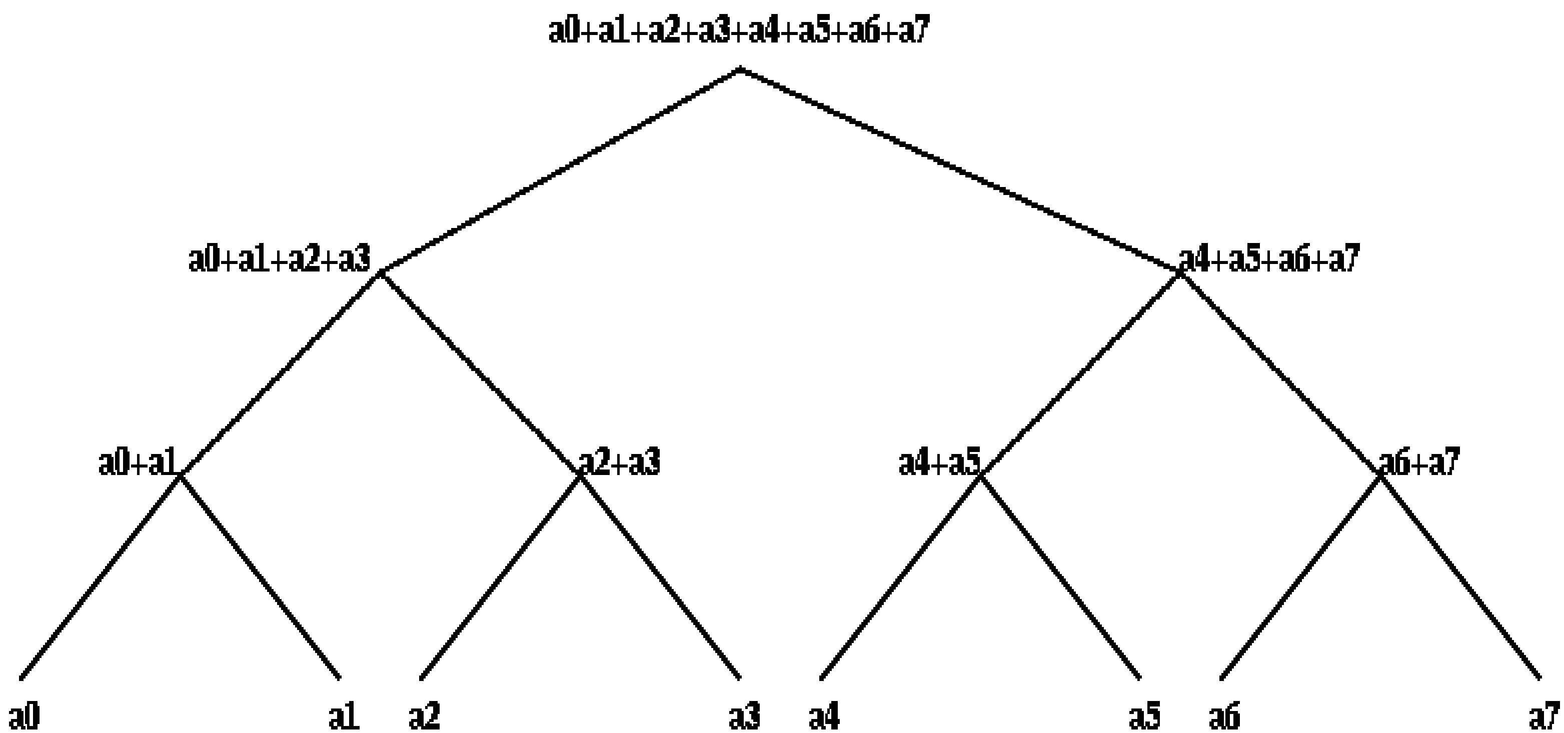
- A log n lower bound to compute any function in parallel
- Reduction and broadcast in O(log n) time
- Parallel prefix (scan) in O(log n) time
- Adding two n-bit integers in O(log n) time
- Multiplying n-by-n matrices in O(log n) time
- Inverting n-by-n triangular matrices in $O(\log^2 n)$ time
- Inverting n-by-n dense matrices in $O(\log^2 n)$ time
- Evaluating arbitrary expressions in O(log n) time
- Evaluating recurrences in O(log n) time
- Solving n-by-n tridiagonal matrices in O(log n) time
- Traversing linked lists
- Computing minimal spanning trees
- Computing convex hulls of point sets

A log n lower bound to compute any function of n variables

- Assume we can only use binary operations, one per time unit
- After 1 time unit, an output can only depend on two inputs
- Use induction to show that after k time units, an output can only depend on 2^k inputs
- A binary tree performs such a computation



Broadcasts and Reductions on Trees



Binary Tree Addition on a Message Passing System

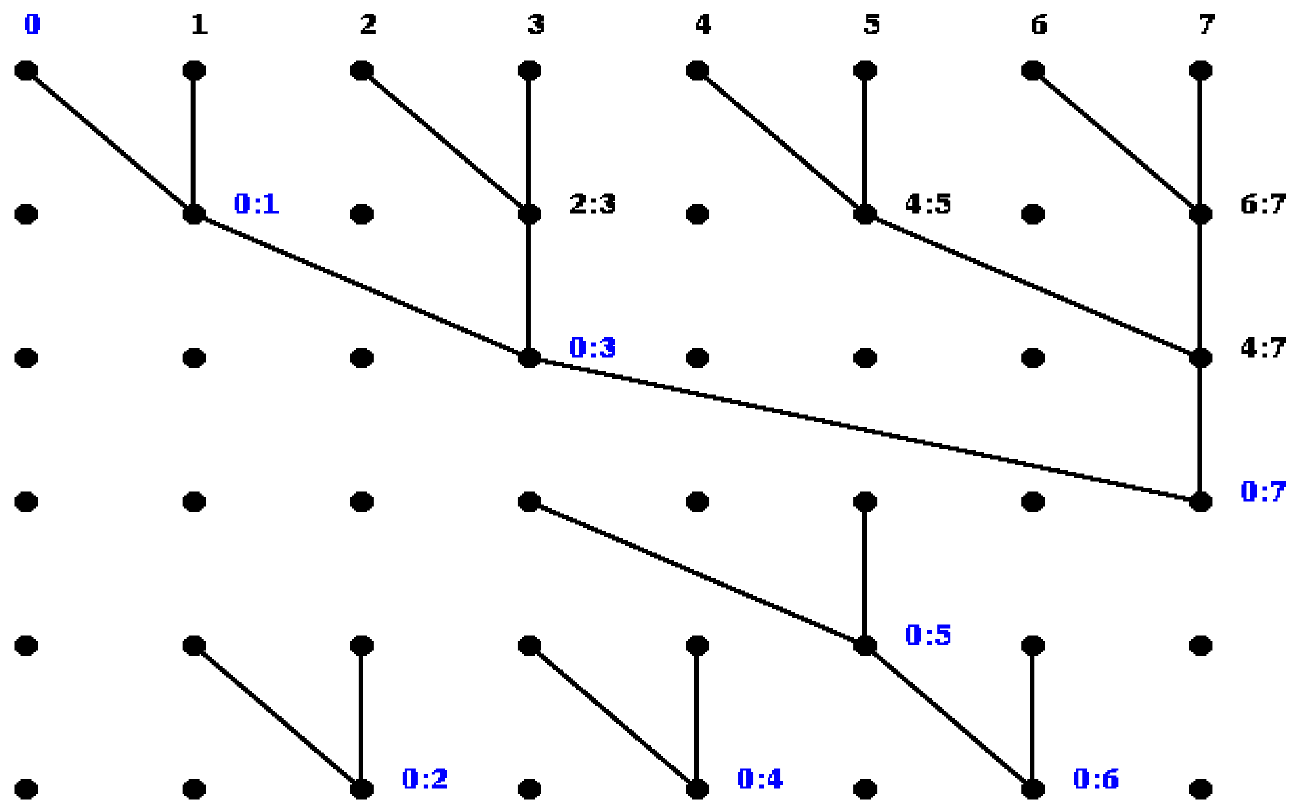
- Suppose we wish to compute the global sum of x_i, contained on processor i. Assume N = 2^n processors.
- Algorithm on processor kp, 0 <= kp < n:
- do for k = 0 to m - 1:
  - Compute ip := ieor (kp, 2^k)
  - Send current x to processor ip.
  - Receive s from processor ip.
  - x := x + s
- enddo
- At completion of loop, processor 0 has global sum.
- This scheme can be easily generalized to non-power-of-two processor counts and to more general arrays.

Parallel Prefix, or Scan

- If "+" is an associative operator, and x[0],…,x[p-1] are input data then parallel prefix operation computes

  y[j] = x[0] + x[1] + … + x[j]   for j=0,1,…,p-1

- Notation: j:k mean x[j]+x[j+1]+…+x[k], blue is final value



Mapping Parallel Prefix onto a Tree - Details

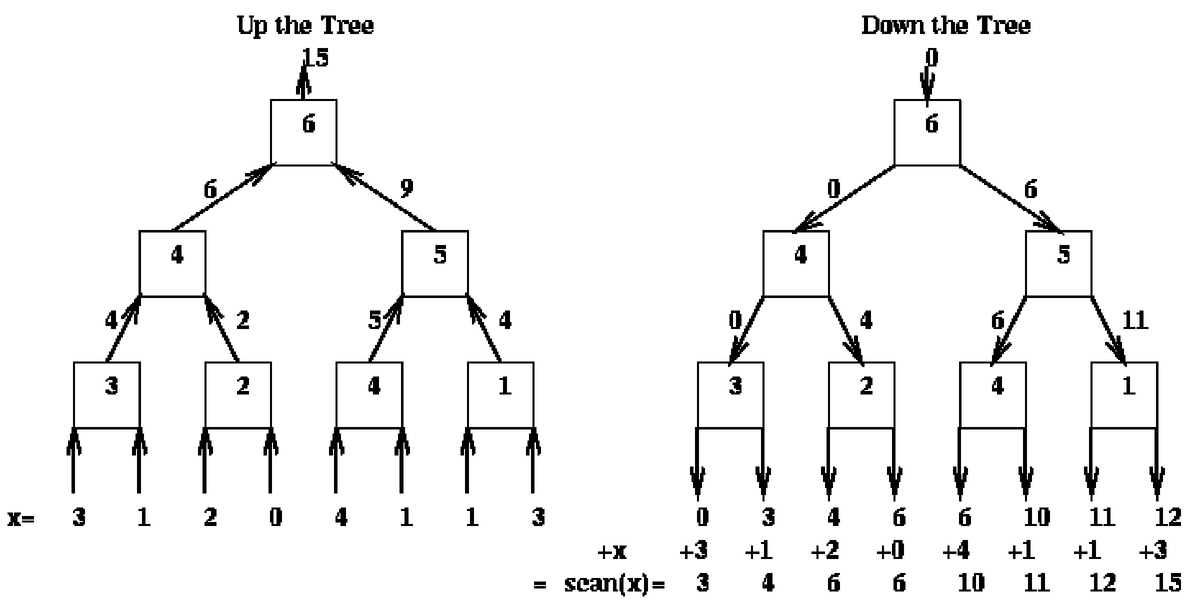
° Up-the-tree phase (from leaves to root)

1) Get values L and R from left and right children
2) Save L in a local register M
3) Pass sum S = L+R to parent

° Down the tree phase (from root to leaves)

1) Get value S from parent (the root gets 0)
2) Send S to the left child
3) Send S + M to the right child

° By induction, S = sum of all leaves to left of subtree rooted at the parent



Adding two n-bit integers in O(log n) time

- **Let a = a[n-1]a[n-2]…a[0] and b = b[n-1]b[n-2]…b[0] be two n-bit binary numbers**
- **We want their sum s = a+b = s[n]s[n-1]…s[0]**

  c[-1] = 0 … rightmost carry bit
  for i = 0 to n-1
  c[i] = ((a[i] xor b[i]) and c[i-1]) or (a[i] and b[i]) ... next carry bit
  s[i] = a[i] xor b[i] xor c[i-1]

- **Challenge: compute all c[i] in O(log n) time via parallel prefix**

  for all (0 <= i <= n-1) p[i] = a[i] xor b[i] … propagate bit
  for all (0 <= i <= n-1) g[i] = a[i] and b[i] … generate bit

$$\begin{bmatrix} c[i] \\ 1 \end{bmatrix} = \begin{bmatrix} (\ p[i] \text{ and } c[i-1]\ ) \text{ or } g[i] \\ 1 \end{bmatrix} = \begin{bmatrix} p[i] & g[i] \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} c[i-1] \\ 1 \end{bmatrix} = C[i] * \begin{bmatrix} c[i-1] \\ 1 \end{bmatrix}$$

… 2-by-2 Boolean matrix multiplication (associative)

$$= C[i] * C[i-1] * \dots C[0] * \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

… evaluate each P[i] = C[i] * C[i-1] * … * C[0] by parallel prefix

- **Used in all computers to implement addition - Carry look-ahead**

Multiplying n-by-n matrices in O(log n) time

- For all (1 <= i,j,k <= n)   P(i,j,k) = A(i,k) * B(k,j)
  - cost = 1 time unit, using n^3 processors
- For all (1 <= I,j <= n) $C(i,j) = \sum_{k=1}^{n} P(i,j,k)$
  - cost = O(log n) time, using a tree with n^3 / 2 processors

Inverting triangular n-by-n matrices in $O(\log^2 n)$ time

- **Fact:** $$\begin{bmatrix} A & 0 \\ C & B \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & 0 \\ -B^{-1}CA^{-1} & B^{-1} \end{bmatrix}$$

- **Function TriInv(T)** … assume $n = \dim(T) = 2^m$ for simplicity

```
If T is 1-by-1
   return 1/T
else
   …  Write T = [ A  0 ]
                [ C  B ]
   In parallel do {
          invA = TriInv(A)
          invB = TriInv(B)  }     …  implicitly uses a tree
   newC = -invB * C * invA
   Return [ invA   0    ]
          [ newC  invB  ]
```

- **time(TriInv(n)) = time(TriInv(n/2)) + O(log(n))**
  - Change variable to $m = \log n$ to get $\text{time(TriInv(n))} = O(\log^2 n)$

Inverting Dense n-by-n matrices in $O(\log^2 n)$ time

- **Lemma 1: Cayley-Hamilton Theorem**
  - expression for A^{-1} via characteristic polynomial in A
- **Lemma 2: Newton's Identities**
  - Triangular system of equations for coefficients of characteristic polynomial
- **Lemma 3:** $\mathrm{trace}(A^k) = \sum_{i=1}^{n} A^k[i,i] = \sum_{i=1}^{n} [\lambda_i(A)]^k$
- **Csanky's Algorithm (1976)**

  1) Compute the powers $A^2, A^3, \ldots, A^{n-1}$ by parallel prefix
     cost = $O(\log^2 n)$
  2) Compute the traces $s_k = \mathrm{trace}(A^k)$
     cost = $O(\log n)$
  3) Solve Newton identities for coefficients of characteristic polynomial
     cost = $O(\log^2 n)$
  4) Evaluate A^{-1} using Cayley-Hamilton Theorem
     cost = $O(\log n)$
- **Completely numerically unstable**

Evaluating arbitrary expressions

- Let E be an arbitrary expression formed from +, -, *, /, parentheses, and n variables, where each appearance of each variable is counted separately
- Can think of E as arbitrary expression tree with n leaves (the variables) and internal nodes labelled by +, -, * and /
- Theorem (Brent): E can be evaluated in O(log n) time, if we reorganize it using laws of commutativity, associativity and distributivity
- Sketch of (modern) proof: evaluate expression tree E greedily by
    - collapsing all leaves into their parents at each time step
    - evaluating all “chains” in E with parallel prefix

Evaluating recurrences

- Let $x_i = f_i(x_{i-1})$, f_i a rational function, x_0 given
- How fast can we compute x_n?
- Theorem (Kung): Suppose degree(f_i) = d for all i
  - If d=1, x_n can be evaluated in O(log n) using parallel prefix
  - If d>1, evaluating x_n takes Ω(n) time, i.e. no speedup is possible
- Sketch of proof when d=1

  $x_i = f_i(x_{i-1}) = (a_i * x_{i-1} + b_i)/(c_i * x_{i-1} + d_i)$ can be written as

  $x_i = num_i / den_i = (a_i * num_{i-1} + b_i * den_{i-1})/(c_i * num_{i-1} + d_i * den_{i-1})$ or

  $$\begin{pmatrix} num_i \\ dem_i \end{pmatrix} = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix} * \begin{pmatrix} num_{i-1} \\ den_{i-1} \end{pmatrix} = M_i * \begin{pmatrix} num_{i-1} \\ den_{i-1} \end{pmatrix} = M_i * M_{i-1} * \ldots * M_1 * \begin{pmatrix} num_0 \\ den_0 \end{pmatrix}$$

  Can use parallel prefix with 2-by-2 matrix multiplication

- Sketch of proof when d>1
  - degree(x_i) as a function of x_0 is d^i
  - After k parallel steps, degree(anything) <= 2^k
  - Computing x_i take Ω(i) steps

Summary of tree algorithms

- Lots of problems can be done quickly - in theory - using trees
- Some algorithms are widely used
  - broadcasts, reductions, parallel prefix
  - carry look ahead addition
- Some are of theoretical interest only
  - Csanky’s method for matrix inversion
  - Solving general tridiagonals (without pivoting)
  - Both numerically unstable
  - Csanky needs too many processors
- Embedded in various systems
  - CM-5 hardware control network
  - MPI, Split-C, Titanium, NESL, other languages